Practica 2 TSI

José Manuel Pérez Lendinez

May 23, 2019

Contents

1 Ejercicio 1

1.1 Representación del mundo

En este caso he optado una representación muy sencilla basada en los siguientes tipos.:

1. Zonas: En este caso representan las partes de mi mapa. Y tienen la siguiente representación.

$$z1\ z2\ ...\ zn - zona$$

2. orientacion: Utilizada para saber hacia donde esta mirando el jugador y para saber que como se conectan las zonas. Su representación es :

$$norte\ sur\ este\ oeste\ -\ orientacion$$

3. Objetos y Personajes: En este caso se tendrá una linea por cada objeto o personaje en el fichero del problema que tendrá el nombre del objeto y el tipo. Como los siguientes ejemplos:

$$princesa1\ -\ princesa$$

$$princesa2\ -\ princesa$$

$$oro1\ -\ oro$$

   Para poder referenciarlos en el dominio de forma mas sencilla añadiremos los siguientes tipos:

$$princesa\ principe\ bruja\ profesor\ dicaprio\ -\ personaje$$

$$oscar\ manzana\ rosa\ algoritmo\ oro\ -\ objeto$$

4. Jugador: En este caso tendremos un tipo especifico para los jugadores como mostrare a continuación:

$$jugador1\ -\ jugador$$

$$jugador2\ -\ jugador$$

Para representar las conexiones entre zonas tengo la siguiente estructura, donde indico que esta conectado y en que punto cardinal.

$$(conectada\ norte\ z13\ z14)$$

$$(conectada\ sur\ z14\ z13)$$

Lo hago con una doble direccinalidad porque a la hora de moverme es mas sencillo buscar directamente desde la actual hasta la de destino que estar comprobando si esta guardada de una manera o otra con when. Tenemos que representar la posición que tomara un elemento en el mundo. Estos elementos pueden ser los jugadores, objetos y personajes. Para esto utilizo la siguiente estructura.

$$(posicion\_jugador\ jugador1\ z1)$$

$$(posicion\_personaje\ bruja1\ z5)$$

$$(posicion\_objeto\ oro1\ z5)$$

Lo único que nos queda por representar es la orientación del jugador que nos dirá si puede cambiar a una zona. Esto se realiza mediante el siguiente ejemplo:

$$(orientacion\_jugador\ jugador1\ norte)$$

1.2 Funcionalidad

En este caso se nos pide que:

1. Girar a izquierda: En este caso tendremos que tener en cuenta la orientación del jugador. De forma que si esta orientado al norte y gira a la izquierda su orientación cambiaría a oeste. Por tanto nos hará falta unicamente el parámetro ?player que sera de tipo jugador y mediante when y el registro de orientación del jugador se cambiara la orientación de este. Un ejemplo seria el siguiente.

   Tenemos el ?player mirando al norte. Para girarlo necesitaríamos un when que si se cumple que esta mirando al norte borre esa orientación y escriba la orientación al oeste para ?player. Este when se tendría que realizar con las 4 posibles orientaciones.

```
(:action girarDerecha
:parameters (?player - jugador)
:effect
(and
        (when (orientacion_jugador ?player norte)
                (and
                        (not(orientacion_jugador ?player norte))
                        (orientacion_jugador ?player este)
                )
        )
        (when (orientacion_jugador ?player este)
                (and
                        (not(orientacion_jugador ?player este))
                        (orientacion_jugador ?player sur)
                )
        )
        (when (orientacion_jugador ?player sur)
                (and
                        (not(orientacion_jugador ?player sur))
                        (orientacion_jugador ?player oeste)
                )
        )
        (when (orientacion_jugador ?player oeste)
```

```
                (and
                        (not(orientacion_jugador ?player oeste))
                        (orientacion_jugador ?player norte)
                )
        )
))
```

2. Girar a derecha: En este caso seria igual que el ejemplo anterior unicamente que si esta orientado por ejemplo al norte pasaría al este. La función girar a la derecha es prácticamente igual unicamente cambiando eso. Por esto no la pondré en la memoria.

3. Coger objeto: A la hora de coger un objeto hay que tener en cuenta que el jugador que lo coja y el objeto tienen que estar en la misma zona. Ademas tendremos que eliminar de la zona dicho objeto para que no pueda cogerse mas veces. Para esto como parámetros necesitaremos como hemos dicho el jugador, el objeto y la zona en la que se situan.

```
:parameters     (?player - jugador ?objeto - objeto, ?zona - zona)
```

   Con esto tendremos que comprobar que en la ?zona este el ?objeto y el ?jugador al mismo tiempo.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (posicion_objeto ?objeto ?zona)
)
```

   Con esto ya sabemos que estamos en la posición indicada tanto el jugador como el objeto. Por tanto tendremos que eliminar (posicion_objeto ?objeto ?zona) e indicar que el objeto lo tiene el ?payer. Para esto tendremos que añadir un nuevo predicado que indicara que el jugador tiene un objeto. El predicado seria el siguiente.

$$(jugador\_tiene\ ?player\ -\ jugador\ ?objeto\ -\ objeto)$$

   Con esto ya se puede realizar la opción de coger objeto.

```
:effect(and
        (not (posicion_objeto ?objeto ?zona))
        (jugador_tiene ?player ?objeto)
)
```

   En este caso como el ejercicio no indica en ningún momento que no se puedan coger mas de un objeto a la vez no he limitado esto. Aunque mas adelante esto si se especifica en otro ejercicio.

4. Dejar objeto: La mecánica de dejar objeto es muy parecida a la anterior. Tiene los mismos parámetros. Puesto que nos hace falta saber el ?player que soltara el objeto, el objeto que soltara y la zona en la que sera soltado. En este caso las precondiciones tiene que cumplir que el jugador tenga el objeto y que esta en la zona donde vamos a soltarlo.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (jugador_tiene ?player ?objeto)
)
```

A la hora de soltarlo tendremos que hacer justo lo contrario que en el caso de coger. Marcaremos que el jugador ya no tiene el objeto y que el objeto se encuentra en la zona.

```
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (posicion_objeto ?objeto ?zona)
)
```

5. Moverse entre zonas: En este caso para movernos entre zonas se tiene que cumplir que estemos mirando a la orientación correcta. Por ejemplo si estamos en z1 y esta conectada con z2 por el norte. Solo podremos pasar de z1 a z2 si el jugador esta orientado al norte. Para esto necesitaremos como parámetros la zona de destino y origen, la orientacion del jugador y el propio jugador.

```
:parameters(
        ?player - jugador
        ?cardinal - orientacion
        ?zona_actual - zona
        ?zona_destino - zona
)
```

Para poder hacer el movimiento necesitaremos estar en la ?zona_actual y orientados en la misma dirección que la ?zona_destino.

```
:precondition (and
        (orientacion_jugador ?player ?cardinal)
        (posicion_jugador ?player ?zona_actual)
        (conectada ?cardinal ?zona_actual ?zona_destino)
)
```

Si se cumplen estas precondiciones podremos movernos, para esto necesitaremos se elimine la posición actual de jugador y se cambie por la ?zona_destino.

```
:effect(and
        (not (posicion_jugador ?player ?zona_actual))
        (posicion_jugador ?player ?zona_destino)

)
```

2 Ejercicio 2

2.1 Representación del mundo

El ejercicio nos pide añadir costes a los caminos entre zonas. De forma que moverse de z1 a z2 tenga un coste x. Para esto necesitaremos añadir nuevos predicados que nos informen del coste entre las zonas. Para esto he optado por añadir un nuevo predicado de tipo función donde se almacenara el coste del camino entre dos zonas. La estructura es la siguiente:

$$(= \ (coste \ z4 \ z3) \ 1)(= \ (coste \ z3 \ z4) \ 1)$$

Esto identifica las dos zonas y el coste es el 3 parámetro. En este caso pasar de la zona 4 a la zona 3 tendría un coste de 1. Uso la doble direccionalidad también para ahorrarme comprobaciones al mirar los costes.

También tendremos que añadir un nuevo predicado de tipo función para llevar la cuenta del coste total.

$$(= \ (coste\_total) \ 0)$$

2.2 Funcionalidad

En este caso los cambios que se realizan son muy pequeños. Tenemos que ir sumando el coste del arco a la hora de movernos con la función para moverse. Por tanto solo tendríamos que añadir a esta función la linea que nos incremente el coste total. La funcino quedaria de la siguiente manera.

```
:effect(and
        (not (posicion_jugador ?player ?zona_actual))
        (posicion_jugador ?player ?zona_destino)
        (increase (coste_total) (coste ?zona_actual ?zona_destino))

)
```

3 Ejercicio 3

3.1 Representación del mundo